Network

Set up the default host name of the machine as well as `localhost`:

> **NOTE:** `sebin-desktop` is used as an example here. Set `$HOSTNAME` to whatever you like.

```
# Define an environment variable containing the desired hostname
export HOSTNAME='sebin-desktop'

# Set the hostname of the machine
echo "$HOSTNAME" > /etc/hostname

# Set localhost to resolve to the machine's loopback address
echo "127.0.0.1□localhost" >> /etc/hosts
echo "::1□□localhost" >> /etc/hosts
```

Set wireless region

If your machine has Wi-Fi it is advisable to set the region for wireless radio waves to comply with local regulations. Not doing this will limit you to 2.4 GHz Wi-Fi, which will likely under-utilize the Wi-Fi bandwidth on your device.

Install `wireless-regdb` for utilities:

```
pacman -S wireless-regdb
```

To set your region temporarily, e.g. Germany:

```
iw reg set DE
```

To set it permanently, uncomment the line with your country in the file `/etc/conf.d/wireless-regdom`. Remember to rebuild your initramfs with `mkinitcpio -P` to apply the changes when the system boots.

Network manager

Previously we installed NetworkManager as our default network mangaging software. GNOME and KDE have out of the box support for managing network connections in their settings dialogs in a graphical manner. Both rely on NetworkManager.

Enable NetworkManager to start at boot:

```
systemctl enable NetworkManager
```

Using `iwd` as the Wi-Fi backend (optional)

By default NetworkManager uses `wpa_supplicant` for managing Wi-Fi connections.

`iwd` (iNet wireless daemon) is a wireless daemon for Linux written by Intel. The core goal of the project is to optimize resource utilization by not depending on any external libraries and instead utilizing features provided by the Linux Kernel to the maximum extent possible.

To enable the experimental iwd backend, first install `iwd` and then create a configuration file:

```
pacman -S iwd
nano /etc/NetworkManager/conf.d/wifi_backend.conf
```

Set the following in the configuration file:

```
[device]
wifi.backend=iwd
```

When NetworkManager starts, it will now use `iwd` for managing wireless connections.

IPv6 Privacy Extensions

By default, Arch enables IPv6, but with the actual public IP address exposed. IPv6 includes the MAC address of the network interface. IPv6 Privacy Extensions mangle the public IP address in a way that prevents the actual address from being known publicly.

Enabling IPv6 Privacy Extensions can be done in different ways:

1. via `sysctl` parameters, setting it at the lowest level
2. via NetworkManager

If not set via the global NetworkManager config or a connection profile (i.e. per connection setting), NetworkManager uses sysfs to determine if IPv6 Privacy Extensions should be enabled.

`sysctl`

To enable IPv6 Privacy Extensions via sysfs during boot, `sysctl` parameters in a config file can be used.

There are 3 parameters by which control behavior:

> **NOTE:** The spelling for the parameter `temp_prefered_lft` is not a typo!

Name	Value	Description
`use_tempaddr`	2	0 = disabled, 1 = enable, prefer real IP, 2 = enable, prefer temporary IP
`temp_prefered_lft`	86400	Preferred life time of temporary IP in seconds (default = 1 day)
`temp_valid_lft`	604800	Maximum life time of temporary IP in seconds (default = 7 days)

These parameters can be applied to:

1. set the parameter on `all` connections
2. set the parameter on the `default` connection
3. set the parameter on a specific network interface (`nic`)

Create a config file such as `/etc/sysctl.d/40-ipv6.conf` and choose your parameter values for one of the three ways of setting up IPv6 Privacy extensions.

For **all** network interfaces:

```
# Enable IPv6 Privacy Extensions
net.ipv6.conf.all.use_tempaddr = 2
net.ipv6.conf.all.temp_prefered_lft = 86400
net.ipv6.conf.all.temp_valid_lft = 604800
```

For the **default** network interface:

```
# Enable IPv6 Privacy Extensions
net.ipv6.conf.default.use_tempaddr = 2
net.ipv6.conf.default.temp_prefered_lft = 86400
net.ipv6.conf.default.temp_valid_lft = 604800
```

For a specific network interface, e.g. the first Wi-Fi adapter called **wlan0**:

```
# Enable IPv6 Privacy Extensions
net.ipv6.conf.wlan0.use_tempaddr = 2
net.ipv6.conf.wlan0.temp_prefered_lft = 86400
net.ipv6.conf.wlan0.temp_valid_lft = 604800
```

NetworkManager

> **NOTE:** If you set up IPv6 Privacy Extensions via `sysctl` config, NetworkManager will use it automatically.

NetworkManager can be set up to enable IPv6 Privacy Extensions. This can either be done globally or per connection profile.

To enable it **globally** create the config file `/etc/NetworkManager/conf.d/ip6-privacy.conf` with the following contents:

```
[connection]
ipv6.ip6-privacy=2
```

This will apply the setting across all current and future connections.

To enable it only **for specific connections**, open the connection profile, e.g. `/etc/NetworkManager/system-connections/<connection name>.nmconnection`, look for the `[ipv6]` section in the file and add the following:

```
...
[ipv6]
...
ip6-privacy=2
...
```

Connection profile files are named the same as their corresponding network, so `Wired Connection 1.nmconnection` or the name of any Wi-Fi network you ever connected to. When you connect to a new network, you will have to apply these settings again for the new connection.

`systemd-resolved` for DNS name resolution

`systemd-resolved` is a `systemd` service that provides network name resolution to local applications via a D-Bus interface, the `resolve` NSS service, and a local DNS stub listener on `127.0.0.53`.

Benefits of using `systemd-resolved` include:

- `resolvectl` as the primary single command for interfacing with the network name resolver service
- A system-wide DNS cache for speeding up subsequent name resolution requests
- Split DNS when using VPNs, which can help in preventing DNS leaks when connecting to multiple VPNs (See Fedora Wiki for a detailed explenation why this is important)
- Integrated DNSSEC capabilities to verify the authenticity and integrity of name resolution requests, e.g. to prevent cache poisoning/DNS hijacking
- DNS over TLS for further securing name resolution requests by encrypting them, improving privacy (not to be confused with DNS over HTTPS)

To use `systemd-resolved` enable the respective unit:

```
systemctl enable systemd-resolved
```

To provide domain name resolution for software that reads `/etc/resolv.conf` directly, such as web browsers and GnuPG, `systemd-resolved` has four different modes for handling the file

- **stub:** a symlink to the `systemd-resolved` managed file `/run/systemd/resolve/stub-resolv.conf` containing only the stub resolver and search domains
- **static:** a symlink to the static `systemd-resolved` owned file `/usr/lib/systemd/resolv.conf` containing only the stub resolver, but no search domains
- **uplink:** a symlink to the `systemd-resolved` managed file `/run/systemd/resolve/resolv.conf` containing all upstream DNS servers known to `systemd-resolved`, effectively bypassing the stub resolver
- **foreign:** an external tool managing system-wide DNS entries for `systemd-resolved` to derive its DNS configuration from

The recommended mode is *stub.*

> **ATTENTION:** A few notes about setting this up:
>
> - Failure to properly configure `/etc/resolv.conf` will result in broken DNS resolution!

- Attempting to symlink `/etc/resolv.conf` whilst inside `arch-chroot` will not be possible, since the file is bind-mounted from the archiso live system. In this case, create the symlink from *outside* `arch-chroot`:

  ```
  ln -sf ../run/systemd/resolve/stub-resolv.conf /mnt/etc/resolv.conf
  ```

- Some DHCP and VPN clients use the `resolvconf` program to set name server and search domains (see this list). For these, you also need to install the `systemd-resolvconf` package to provide a `/usr/bin/resolvconf` symlink.

This propagates the `systemd-resolved` managed configuration to all clients. To use it, replace `/etc/resolv.conf` with a symbolic link to it:

```
ln -sf ../run/systemd/resolve/stub-resolv.conf /etc/resolv.conf
```

When set up this way, NetworkManager automatically picks up `systemd-resolved` for network name resolution.

Fallback DNS servers

If `systemd-resolved` does not receive DNS server addresses from the network manager and no DNS servers are configured manually, then `systemd-resolved` falls back to a hardcoded list of DNS servers.

The fallback order is:

1. Cloudflare
2. Quad9 (without filtering and without DNSSEC)
3. Google

> **ATTENTION:** Depending on your use-case, you might not want to route all your DNS traffic through the pre-determined fallback servers for privacy reasons. Do your own research on fallback DNS servers that you want to trust.

Fallback addresses can be manually set in a drop-in config file, e.g. `/etc/systemd/resolved.conf.d/fallback_dns.conf`:

```
[Resolve]
FallbackDNS=127.0.0.1 ::1
```

To disable the fallback DNS functionality set the `FallbackDNS` option without specifying any addresses:

```
[Resolve]
FallbackDNS=
```

DNSSEC

> **WARNING:** DNSSEC support in `systemd-resolved` is considered experimental and incomplete .

DNSSEC is an extension to the DNS system that verifies DNS entries via authentification and data integrity checks to prevent DNS cache poisoning, but *does not **encrypt** DNS queries*. For actually encrypting your DNS traffic, see the section below.

`systemd-resolved` can be configured to use DNSSEC for validation of DNS requests. It can be configured in three modes:

Setting	Description
`allow-downgrade`	Validate DNSSEC only if the upstream DNS server supports it
`true`	Always validate DNSSEC, breaking DNS resolution if the server does not support it
`false`	Disable DNSSEC validation entirely

Set up DNSSEC in a drop-in config file, e.g. `/etc/systemd/resolved.conf.d/dnssec.conf`:

```
[Resolve]
DNSSEC=allow-downgrade
```

DNS over TLS

DNS over TLS (DoT) is a security protocol for encrypting DNS queries and responses via Transport Layer Security (TLS), thereby increasing privacy and security by preventing eavesdropping on DNS requests by internet service providers and malicious actors in man-in-the-middle attack scenarios.

DNS over TLS in `systemd-resolved` is disabled by default. To enable validation of your DNS provider's server certificate, include their hostname in the `DNS` setting in the format `ip_address#hostname` and set `DNSOverTLS` to one of three modes:

Setting	Description
`opportunistic`	Attempt DNS over TLS when possible and fall back to unencrypted DNS if the server does not support it

Setting	Description
`true`	Always use DNS over TLS, breaking resolution if the server does not support it
`false`	Disable DNS over TLS entirely

> **ATTENTION:** When setting `DNSOverTLS=opportunistic` `systemd-resolved` will try to use DNS over TLS and if the server does not support it fall back to regular DNS. Note, however, that this opens you to "downgrade" attacks, where an attacker might be able to trigger a downgrade to non-encrypted mode by synthesizinig a response that suggests DNS over TLS was not supported.

> **WARNING:** If setting `DNSOverTLS=yes` and the server provided in `DNS=` does not support DNS over TLS *all DNS requests will fail!*

To enable DNS over TLS system-wide for all connections, add your DNS over TLS capable servers in a drop-in config file, e.g. `/etc/systemd/resolved.conf.d/dns_over_tls.conf`:

```
[Resolve]
DNS=9.9.9.9#dns.quad9.net 149.112.112.112#dns.quad9.net [2620:fe::fe]#dns.quad9.net
[2620:fe::9]#dns.quad9.net
DNSOverTLS=yes
```

Alternatively, you can use drop-in configuration files for NetworkManager to instruct it to use DNS over TLS per connection. You can save this as a drop-in configuration file under `/etc/NetworkManager/conf.d/dns_over_tls.conf` to apply it to current and future connections or on a per-connection basis to an existing connection profile under `/etc/NetworkManager/system-connections/*.nmconnection` (as `root`).

There's three possible values:

- 2 = DNS over TLS always on (fail if DoT is unavailable)
- 1 = opportunistic DNS over TLS (downgrades to unencrypted DNS if DoT is unavailable)
- 0 = never use DNS over TLS

Add or modify

```
[connection]
dns-over-tls=2
```

Multicast DNS

`systemd-resolved` is capable of working as a multicast DNS (mDNS) resolver and responder. The resolver provides hostname resolution using a "*hostname*.local" naming scheme.

mDNS support in `systemd-resolved` is enabled by default. For a given connection, mDNS will only be activated if both mDNS in `systemd-resolved` is enabled, and if the configuration for the currently active network manager enables mDNS for the connection.

The `MulticastDNS` setting in `systemd-resolved` can be set to one of the following:

Setting	Description
`resolve`	Only enables resolution support, but responding is disabled
`true`	Enables full mDNS responder and resolver support
`false`	Disables both mDNS responder and resolver

> **ATTENTION:** If you plan on using `systemd-resolved` as mDNS resolver and responder consider the following:
>
> - Some desktop environments have the `avahi` package as a dependency. To prevent conflicts, `disable` or `mask` both `avahi-daemon.service` and `avahi-daemon.socket`
> - If you plan on using a firewall, make sure UDP port `5353` is open

To enable mDNS for a connection managed by NetworkManager tell `nmcli` to modify an existing connection:

```
nmcli connection modify CONNECTION_NAME connection.mdns yes
```

> **TIP:** The default for all NetworkManager connections can be set by creating a configuration file in `/etc/NetworkManager/conf.d/` and setting `connection.mdns=2` (equivalent to "yes") in the `[connection]` section.
>
> ```
> [connection]
> connection.mdns=2
> ```

Avahi

Avahi implements zero-configuration networking (zeroconf), allowing for multicast DNS/DNS-SD service discovery. This enables programs to publish and discover services and hosts running on a

local network, e.g. network file sharing servers, remote audio devices, network printers, etc.

Some desktop environments pull in the `avahi` package as a dependency. It enables their file manager to scan the network for services and make them easily accessible.

> **ATTENTION:** If you plan on using `avahi` as mDNS resolver and responder consider the following:
>
> - You need to disable mDNS in `systemd-resolved`. You can do so in a drop-in config file, e.g. `/etc/systemd/resolved.conf.d/mdns.conf`:
>
>   ```
>   [Resolve]
>   MulticastDNS=false
>   ```
>
> - If you plan on using a firewall, make sure UDP port `5353` is open

Avahi provides local hostname resolution using a "*hostname*.local" naming scheme. To use it, install the `avahi` and `nss-mdns` package and enable Avahi:

```
pacman -S avahi nss-mdns
systemctl enable avahi-daemon
```

Then, edit the file `/etc/nsswitch.conf` and change the `hosts` line to include `mdns_minimal [NOTFOUND=return]` before `resolve` and `dns`:

```
hosts: mymachines mdns_minimal [NOTFOUND=return] resolve [!UNAVAIL=return] files myhostname
dns
```

To discover services running in your local network:

```
avahi-browse --all --ignore-local --resolve --terminate
```

To query a specific host for the services it advertises:

```
avahi-resolve-host-name hostname.local
```

Avahi also includes the `avahi-discover` graphical utility that lists the various services on your network.

Revision #13
Created 2022-02-12 00:56:17 UTC by Sebin
Updated 2026-02-07 02:38:14 UTC by Sebin